Python code followed by a picture of my by-hand attempt.

```
import numpy
from numpy import *
from numpy.linalg import *

# With numpy matrices, you can add, subtract, multiply, find the transpose
# find the inverse, solve systems of linear equations and much more.

# Solve a system of consistent linear equations. Refer to Lial Section 2.5
# Example 7 Cryptography for the calculation

# Right hand side of system of equations has data entered as a list
# and converted to 3x1 matrix and then a 1x3 matrix using the transpose
# function. Similar steps are taken for the matrix A.

rhs= [10, 6, 5, 7]
rhs=matrix(rhs)
rhs=transpose(rhs)
print ('\nRight Hand Side of Equation')
print rhs

A =[[2,90,2,0], [1,80,2,3],[1,20,3,7],[2,30,0,7]]
A= matrix(A)
print ('\nMatrix A')
print A

# Numpy has various functions to perform matrix calculations. The inverse
# function inv() is one of those.

# Find inverse of A.
print ('\nInverse of A')
IA= inv(A)
print IA

# In what follows, I am converting matrices with floating point numbers to
# matrices with integer numbers. This is optional and being done to show
# that it is possible to do so with numpy matrices.

# Note that the function dot() performs matrix multiplication.
# Verify inverse by multiplying matrix A and its inverse IA.

print ('\nIdentity Matrix')
I= dot(IA,A)
I= (I) # This converts floating point to integer.
print I

# Solve the system of equations and convert to integer values.
# With numpy it is necessary to use dot() for the product.
```

```
result = dot(IA,rhs)
#result = int_(result) # This converts floating point to integer.
#result = rint(result).astype(int)
print ('\nSolution to Problem')
print result

# There is a more efficient way to do this with the linalg.solve() function.

print ('\nIllustration of solution with linalg.solve(,) function')
result2= linalg.solve(float_(A),float_(rhs))
print result2 # This converts floating point to integer.

eq1=result[0,0]*2+result[1,0]*90+2*result[2,0]
eq2=result[0,0]+result[1,0]*80+2*result[2,0]+3*result[3,0]
eq3=result[0,0]+result[1,0]*20+3*result[2,0]+7*result[3,0]
eq4=result[0,0]*2+result[1,0]*30+7*result[3,0]
print '\n'
print ('\n equation 1 output: %d') %eq1
print ('\n equation 2 output: %d') %eq2
print ('\n equation 3 output: %d') %eq3
print ('\n equation 4 output: %d') %eq4
```